Formal Modeling of Synchronization Methods for Concurrent Objects in Ada 95

Ravi K. Gedela, Sol M. Shatz and Haiping Xu
Department of EECS
Concurrent Software System Lab
The University of Illinois at Chicago
R.Gedela@ericsson.com
{shatz, hxu1}@eecs.uic.edu
Purpose of Formal Methods

“The term “formal methods” denotes software development and analysis activities that entail a degree of mathematical rigor. (…) A formal method manipulates a precise mathematical description of a software system for the purpose of establishing that the system does or does not exhibit some property, which is itself-precisely defined.” (Dillon and Sankar, 1997)

Concurrent Program Analysis

• Objects communicate with each other and undesirable situations, such as deadlock or livelock, may occur.
• There are two different types of program fault
  – Unconditional fault
  – Conditional fault
• Automated program analysis is vital for debugging and testing a concurrent program.
Introduction to Petri Net

• “Three-in-one” capability of a Petri net model.
  – Graphical representation
  – Mathematical description
  – Simulation tool

• **Definition:**
  A Petri net is a 4-tuple, $PN = (P, T, F, M_0)$ where
  
  $\begin{align*}
  P &= \{P_1, P_2, \ldots, P_m\} \text{ is a finite set of places;} \\
  T &= \{t_1, t_2, \ldots, t_n\} \text{ is a finite set of transitions;} \\
  F \subseteq (P \times T) \cup (T \times P) \text{ is a set of arcs (flow relation);} \\
  M_0 : P \rightarrow \{0, 1, 2, 3, \ldots\} \text{ is the initial marking.}
  \end{align*}$
Example:
Automated Program Analysis Paradigm
Example: Modeling Rendezvous

Internal Representation

**Task A**
- entry --> wait_ack, ack_entry
- ack_accept, wait_ack --> S

**Task B**
- accept, ack_entry --> S
- end_accept --> ack_accept, S
Motivation for This Work

- To illustrate the possibility of translating advanced features of Ada 95 into Petri net.
- To illustrate analysis capability by using a formal modeling tool.
- To provide a graphic viewpoint of synchronization methods to aid understanding for beginner.
Three Synchronization Methods for Concurrent Objects

• Synchronization is added if and when it is required, by extending the object.
• Synchronization is provided by the base (root) object type.
• Synchronization is provided as a separate protected type and the data is passed as a discriminant.

Method 1: Synchronization Added by Extending the Object

```plaintext
package Object is
    procedure Op1 (O : in out Obj_Type);
    procedure Op2 (O : in out Obj_Type);
    ...
    type Obj_Type is tagged limited record ... end record;
end Object;

package Object.Synchronized is
    type Protected_Type is new Obj_Type with record L: Mutex; end record;
end Object.Synchronized;

protected type Mutex is
    entry Lock; procedure Unlock;
end Mutex;
```
Method 1: Synchronization Added by Extending the Object (continue)

```plaintext
package Object.Synchronized.Extended is
    procedure Op1_ext (O : in out Extended_Protected_Type);
    procedure Op2_ext (O : in out Extended_Protected_Type);

    type Extended_Protected_Type is new Protected_Type
        with record…end record;
end Object.Synchronized.Extended;
```

```plaintext
procedure Op1_ext (O : in out Extended_Protected_Type) is
    begin
        O.L.Lock; Op1 (Obj_Type (O)); O.L.Unlock;
    end Op1_ext;
```
Figure 1. Object.Synchronized.Extended model
potential deadlock problem

package Object.Synchronized is

  procedure Op1 (O : in out Protected_Type);
  procedure Op2 (O : in out Protected_Type);

  ...

  type Protected_Type is new Obj_Type with record L : Mutex; end record;

end Object.Synchronized;

procedure Op1 (O : in out Protected_Type) is
begin
  O.L.Lock; Op1(Obj_Type(O)); O.L.Unlock;
end Op1;
package Object.Synchronized.Extended is

procedure Op1_ext (O : in out Extended_Protected_Type);
procedure Op2_ext (O : in out Extended_Protected_Type);
...
type Extended_Protected_Type is new Protected_Type with record...
  record;
end Object.Synchronized.Extended;

procedure Op1_ext (O : in out Extended_Protected_Type) is
begin
  O.L.Lock; -- pre_processing; Op1(Protected_Type (O));
  -- post_processing; O.L.Unlock;
end Op1;
Figure 2. Object.Synchronized.Extended model
Method 2: Synchronization Provided by the Base Object Type

package Protected_Object is

procedure Class_Wide_Op1 (O: in out Protected_Type'Class);
procedure Class_Wide_Op2 (O: in out Protected_Type'Class);
...
type Protected_Type is abstract tagged limited record  L : Mutex;  end record;

procedure Op1 (O : in out Protected_Type) is abstract;
procedure Op2 (O : in out Protected_Type) is abstract;
end Protected_Object;

procedure Class_Wide_Op1 (O: in out Protected_Type'Class) is
begin
   O.L.Lock;  Op1(O);  -- dispatch to correct operation  O.L.Unlock;
end Class_Wide_Op1;
Method 2: Synchronization Provided by the Base Object Type (continue)

```haskell
package Protected_Object.My_Object is
  type My_Object_Type is new Protected_Type with record … end record;
  procedure Op1 (O : in out My_Object_Type);
  procedure Op2 (O : in out My_Object_Type);
end Protected_Object.My_Object;

package Protected_Object.My_Object.Extended is
  type Extended_Protected_Type is new My_Object_Type with record…end record;
  procedure Op1 (O : in out Extended_Protected_Type);
  procedure Op2 (O : in out Extended_Protected_Type);
end Protected_Object.My_Object.Extended;
```
Method 2: Synchronization Provided by the Base Object Type (continue)

MO: My_Object_Type;       -- represented as color M in the following
Class_Wide_Op1(MO);       -- Petri net model

...

EP: Extended_Protected_Type;    -- represented as color E in the following
Class_Wide_Op1(EP);           -- Petri net model

Notes: We will use colored Petri nets to model this method, where “colored” tokens (or tokens with attributes) are used. In colored Petri nets, a transition becomes enabled when its input places have tokens with attributes that match the inscriptions on the corresponding arcs from the place to the transition.
Figure 3. Protected_Object.My_Object.Extended model
Concluding Comments

- Illustrated the possibility of translating advanced features of Ada 95 into Petri net.
- Shown that behavior analysis, such as deadlock analysis, could be automated by using Petri net reachability analysis.
- Performance analysis is possible by using performance Petri net, such as timed Petri net, stochastic Petri net etc.
Definition of Protected Type: Mutex

protected type Mutex is
  entry Lock;  procedure Unlock;
private
  Release: Boolean := True;
end Mutex;

protected body Mutex is
  entry Lock when Release is
    begin  Release := False;  end Lock;

  procedure Unlock
    begin  Release := True;  end Unlock;
end Mutex;